Dependency Analysis in the HTML5, JavaScript and CSS3 Stack

by

Vasu Gupta

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Computing Studies

Approved May 2014 by the
Graduate Supervisory Committee:

Kevin Gary (Chair)
Ajay Bansal
Timothy Lindquist

ARIZONA STATE UNIVERSITY

May 2014

ABSTRACT

The Internet is transforming its look, in a short span of time we have come very far from black and white web forms with plain buttons to responsive, colorful and appealing user interface elements. With the sudden rise in demand of web applications, developers are making full use of the power of HTML5, JavaScript and CSS3 to cater to their users on various platforms. There was never a need of classifying the ways in which these languages can be interconnected to each other as the size of the front end code base was relatively small and did not involve critical business logic. This thesis focuses on listing and defining all dependencies between HTML5, JavaScript and CSS3 that will help developers better understand the interconnections within these languages. We also explore the present techniques available to a developer to make his code free of dependency related defects. We build a prototype tool, HJCDepend, based on our model, which aims at helping developers discover and remove defects early in the development cycle.

For Mummy and Papa,
with love and gratitude.

ACKNOWLEDGMENTS

I would like to express my thanks to chair of my committee, Dr Kevin Gary. For me he has played the role of a mentor, a teacher, an advisor, an employer, a friend and an avid believer of my capabilities in software. This thesis document would not have been possible without his expertise in software engineering, his valuable time, his advice, his criticism and reviews from the beginning to the end. I would also like to thank the members of my committee Dr Timothy Lindquist and Dr. Ajay Bansal for their valuable guidance and support on this thesis and research.

I would like to thank all the student developers who took part in my research study. Many thanks to my family and friends who supported me and helped me complete this research.

TABLE OF CONTENTS

Page

CHAPTER Page

CHAPTER Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

With the sudden rise in demand of web and mobile device applications, developers are making full use of the power of HTML5, JavaScript and CSS3 to cater to their users on various platforms. The original intent of JavaScript and CSS was to enhance the overall user experience, however, these languages are now used by developers to handle business logic of web applications. Elements of these three languages are often tightly coupled to serve functionality. This makes the front end code of even small and mid-sized size applications complex to understand. There is no existing model that defines and lists all the types of interconnections or dependencies that exist between HTML5, JavaScript and CSS3. Developers do not have tools that can do dependency analysis of their code and therefore it becomes difficult for the developer to identify dependency related defects in early phases of development.

HTML5 gives a structure to the web page whose elements are made responsive to user interactions using JavaScript and CSS3 respectively. Unlike statically typed languages like Java, the developer has no way to identify all the dependencies in his code and he makes assumptions, for example:

- All his source files will be available where he expects them to be.
- The return type of a JavaScript function is what he expects it to be.
- A styling class that his JavaScript listener function would apply to a DOM element in response to a user action is present in the CSS file.
- An event listener that he embeds in his DOM element is available in one of the JavaScript files that he included in the HTML file.

The extensive use of these technologies in web applications to implement business logic has risen recently and there has been not much formal effort or research to address these issues faced by developers. Therefore a developer or an architect cannot say with confidence the right way to develop the front end code of their application.

A developer creates a dependency in a file if one component in the front end code requires another component in the front end code to complete its functionality, the definition of dependency is further explained in chapter 3. This thesis focuses on modeling the types of dependencies amongst HTML5, JavaScript and CSS to provide a framework for understanding the interconnections of elements in a web page also it will give the developer an understanding of the extent to which his front end code is coupled. A developer could save time by having access to tools and techniques that look into his code and tell him where potentially his code might be bug ridden. Using the model defined in this thesis we build a prototype, HJCDepend, that helps developers find these dependencies in their code. Web development teams prefer to develop their code in a Rapid Application Development environment [24]. In this kind of environment the client is also involved during the development phase of the application [25]. This way developers maintain a lower cost to fulfill change requests. Therefore the entire development process benefits from the tool because it analyzes the code of the developer and report bugs as he writes code. The implementation of our model tackles one of the many problems which enhance the efficiency of a developer.

This research is unique because the existing research community defined models are for static analysis within these languages; none of them focuses on dependencies among these languages. This research benefits developers to find where defects are injected more readily and identify potential refactoring. It reduces the time frame between defect injection and defect discovery. It would also help other teams associated with the project, for example:

- The deployment team can make sure that their final build has all the required source files using this model.
- Architects can use this model to understand the modularity of the code base by knowing all the connections between the three languages.

Due to the different nature of HTML5, JavaScript and CSS (discussed in chapter 2, section 2.1 and 2.2), we cannot directly import an existing dependency management model from the software world. Most other languages that work together to serve functionality have existing models that developers can rely upon, these models tell them how coupling in different languages should be handled. For example, developers make use of Object Relational Mapping (ORM) in Java and SQL to serve data to user from database. ORM tool like Hibernate[1] acts as a neat intermediate layer to keep track of dependencies between Java and SQL. Also developers leverage from the fact that Java is compiled before it is run, making it easier for them to identify bugs before actually running their code. Java and SQL are languages written to perform specific tasks whereas JavaScript was originally written to do small tasks like tying HTML pages to Java programs. JavaScript was never intended to do heavy weight work of making ajax[2] requests to a server and then present the received response by manipulating the DOM.

[1] http://hibernate.org/
[2] http://en.wikipedia.org/wiki/Ajax_(programming)

Also none of the three languages are compiled making it more difficult for any IDE to identify issues while the developer writes code. The only way for the developer to identify bugs is to put the code on a server and run it on a browser. Their problems do not end there; they have no standard tools to debug faulty code. They have to eventually resort to trial and error means of inspecting each element of the three languages in their code to find the bug. For a production quality application the total front end code could span over multiple files making this method of trial and error technique of finding defects tedious and requiring more than necessary time.

Chapter 2 discusses the research in this field and in several sections of that chapter we bring into light where the current research stands and why the classification of dependencies is necessary. Chapter 3 presents the dependency model and discusses in detail different dependencies that can be created in HTML5, JavaScript and CSS. It also describes how unsatisfied dependencies can affect the application. Chapter 4 describes the implementation of the model defined in chapter 3 into a tool, HJCDepend.

In chapter 5, we validate the model by presenting the results of a research study we conducted. We asked a group of developers to use HJCDepend to debug a body of code that was seeded with dependency related defects. We compare the time they take to find bugs with the tool and without the tool. This research study gives a measure of effectiveness of the model and how the tool can benefit developers while writing code. In chapter 6 we summarize the contributions of this work and future work.

CHAPTER 2

LITERATURE REVIEW

The research in front end technologies HTML5, JavaScript and CSS is at a very nascent stage. The expected functionality for an application from the front end code is constantly growing and one can expect a plethora of solutions on the Internet for any problem a developer faces. Being a technology that has recently caught the attention of developers, the research in this field is not able to keep up with pace of growth of the industry. This literature review presents existing research on dependency analysis in front end technologies.

We could not find many direct attempts to address the issues we are facing but rather we found a very few research papers that tangentially address these issues. Also, there are a large number of frameworks that try to solve these issues but without defining a concrete model. For example, [5], [6] and [7] talk only about static analysis within JavaScript and frameworks like Requirejs helps a developer fetch all JavaScript files that are required in the HTML page. Using these kinds of frameworks a developer would not have to worry about what JavaScript files he needs to bundle when he deploys the application on production environment. The research and framework solutions majorly focus on JavaScript related code, they do not provide developers the tools and techniques that can help them identify bugs in their entire front end code. We discuss why we need dependency management and more about the community ideas to solve this problem in our following sections.

2.1 Cohesion and Coupling in Software

The concepts of cohesion and coupling are not new to the software industry [8], [9] and [10]. Cohesion is defined as the measure of the extent to which the various functions performed by an entity are related to one another [9] and coupling is defined as, the degree of interaction between classes [9]. Making the code concise, maintainable and reusable is one of the important technical challenges every software development team faces [10]. Having code that conforms to the concepts of cohesion and coupling not only reduces the total development time by making it reusable but also makes it easier to modify after its development phase [9]. To achieve this a developer places his related code close to each other and makes modifications to independent code without the fear of breaking functionality. There has been significant research to achieve reusability and maintainability for languages like Java which are built on Object Oriented Programming (OOP) concepts [11] and [12]. Tools like JDepend [13] help a developer find metrics like Afferent Coupling (Ca) [13] and Efferent Coupling (Ce) [13] in their Java code which ultimately help him define how maintainable or reusable his code is. Also writing code in Java or similar other statically typed languages force you to conform to a hierarchical structure; you have packages and in them you have classes. This hierarchical pattern allows developers to reach some form of maintainability and reusability using Object Oriented Programming (OOP). JavaScript does not provide a specific structure for the programmers to follow. Instead a developer has to decide how he wants to structure the code for a web page comprising of HTML5, JavaScript and CSS. These three languages do not possess features that make a developer comply to a structure or hierarchy while writing code. This research does not give a solution to how the front end code should be

structured but the model helps the developer understand how the three languages can be interconnected and how it can help him identify the broken interconnections in the code.

2.2 Dependency Structure Matrix (or Design Structure Matrix)

Engineers tend to decompose their system into smaller sub systems and find relationships between them to understand it better [1]. A common approach to resolve this is to represent everything you know about a system in a tabular format to make the complex system intuitive and easy to understand. A well-known example of representing systems in a tabular format is Dependency Structure Matrix (DSM) [2]. It is a matrix which helps civil, automobile and software engineers to study the systems better by *"decomposing and integrating"* the system according to [1]. DSM in most cases is N^2 matrix where rows and columns are the same set of elements in the system in the same order (figure 1). The diagonal, where same elements meet is shaded and places on either side of the diagonal are marked with an "X". If you read from left to right then an "X" would represent what other elements depend on that element and reading from top to bottom would tell you what elements that element is dependent upon [1].

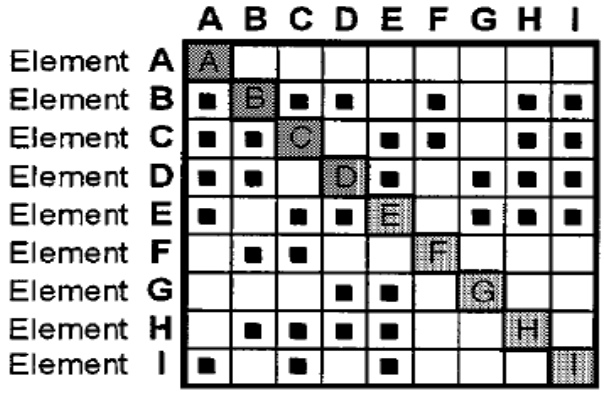

Figure 1 Example DSM [1]

An implementation of DSM is Lattix Inc's Dependency Manager (LDM)[3]. *"It is a tool for management of inter-module dependencies in a software, it uses a standard notion of dependency, in which a module A depends on a module B if there are explicit references in A to syntactic elements of B"* [4]. If a Java code is fed to LDM, it would create a DSM where the rows and columns would be java packages. The smallest subsystem in LDM is a Java class. A user feeds the tool with "design rules" against which the code would be checked during the phase of its development [4]. Deviation at any point of development would be flagged by the tool, this helps the developer keep his related code highly cohesive and independent code loosely coupled.

It would be difficult to apply this technique directly to HTML5, JavaScript and CSS3 stack because there is no module definition or any structure that these languages conform to together. To study these dependencies, in our model we do not subdivide each element, we assume the smallest subsystem to be the three languages themselves.

2.3 JavaScript Complications

Our goal is to find the right balance between static analyses but sticking to mainstream cases that impact the typical developer's situation. We prefer understanding the language features impacting maintainability (and understandability) of JavaScript programs, which may require a more empirical approach. [20] is an example in that direction, scoped to ajax.

[3] http://lattix.com/

Richard et al. [19] claim that the dynamic features of JavaScript like eval which can create code on runtime and function variadicity makes it difficult to do a static analysis on the applications built using it. They show that a large community of developers use these features and exploit them. These dynamic features cannot be evaluated unless the code is deployed on a server and user interacts with the application. Function variadicity, is passing more or less arguments than indicated in the declaration of the function. It is for the developer to make sure that the arguments that he did not pass or the unnecessary arguments that he passed to a JavaScript function do not affect the overall result.

HTML code snippet:

```
<ul>
<li onclick="myFunction(this, "First Item", 1, true)">First Item</li>
//This is valid
<li onclick="myFunction(this)">Second Item</li> //Even this is valid
<ul>
```

JavaScript code snippet:

```
myFunction(callerDOMObj, whichItem){
      …code body…
}
```

Code block 2.1: Example of Function Variadicity

In the HTML snippet above you will notice that the `onclick` listeners on the `li` elements call `myFunction` with different number of parameters and JavaScript would accept such code without throwing any errors. This feature is exploited by developers to achieve some sort of polymorphism but this technique can lead to dangerous situations like trying to access a non existing JavaScript object in the function body. Therefore, in our prototype, we classify missing dependencies as either an error or a warning. This

classification is required so that even if the code works, the developer would know the dangerous practice he is using.

2.4 Dependency Management within JavaScript

JavaScript can have dependencies within itself. For example when a JavaScript function is called from another JavaScript function, then the caller function is dependent upon the function it calls to complete its execution and the caller function requires the HTML to include the JavaScript file that contains the called function. Consequently, a developer should not only make sure that all the JavaScript files that are referenced by the HTML webpage should be included in his web page. Also if a HTML page does not reference any elements within a JavaScript file directly but does indirectly through some other JavaScript file, that file should be included as well.

We previously mentioned that there has not been much formal research in this area, developers have written solutions to the issues they encounter every day. We cannot cite a research paper or journal for our following discussion of RequireJS. However we direct our readers to the source of our findings [14] and [15]. RequireJS is a module loader/ a JavaScript file that helps a developer manage inter JavaScript dependencies and is built on Asynchronous Module Definition specification, i.e. code is written in modular fashion and dependencies are taken care of by RequireJS [23]. A developer, in his JavaScript code, could tell RequireJS to load all the scripts that a function may need while execution to be included in the webpage. Doing this relieves the developer from writing all the script tags and maintaining their right order in the HTML page. However

the drawback of this technique of dependency management is that, a developer is still responsible for specifying the dependencies in his configuration function. There is no way for him to check his code for missing dependencies. Following is a list of similar solutions to JavaScript dependency management problem.

- RequireJS http://requirejs.org/
- Npm https://www.npmjs.org/
- uRequire http://urequire.org/
- Mantri http://mantrijs.com/
- RaptorJS http://raptorjs.org/
- Jam http://jamjs.org/
- Bower http://bower.io/

Most of them give an informal description of the problem itself, and no tools or means by which a developer may identify the extent of the problem. They are not scientifically defined and seem more like random developer community solutions as they explore the problem space. In our model we consider a JavaScript file dependent on another JavaScript file if there is a direct reference of an element from one file to another.

2.5 Community Solutions to Loosely Coupled Front End

Obviel is a JavaScript framework that allows developers to build web applications with loose coupling [17]. Using the Obviel API developers can build components of their view in JavaScript and then connect them together to display them into a complete webpage. The API also facilitates bootstrapping single page application from one URL [17]. Backbone.JS is a similar framework which helps developers the most of the web application in JavaScript by creating small modules. It helps developers separate concerns and structure their code in MVC architecture [18]. We feel that the drawback of

these frameworks is that the developer would have to come out of his comfort zone and code according to Obviel or BackBone.JS specific API. In our model we do not want the developer to change his code structure in any way, we help him manage and understand dependencies he creates himself.

2.6 Type and Static Program Analysis in JavaScript

Research papers such as [3], [5], [6], [7], [16] and [26] explore type analysis, flow analysis and static program analysis in JavaScript. [6] explores taint analysis for JavaScript static analysis and talks about certain language features. Also, like most static analysis papers [26] focuses on security-related defects, but not always in the same category as software maintenance and evolution. These papers, for example, focus on points-to analysis (figuring out in a dynamic language what a reference actually points to, as it can be dynamically computed or assigned), call-graph analysis (you can dynamically determine functions to call at run-time based on property or variable assignment), and in general reflective characteristics (e.g. eval) of the language - useful since JavaScript libraries like JQuery make heavy use of reflection.

The peculiarities of JavaScript when compared to other languages make program and static analysis difficult [3]. For example there are situations when JavaScript's type conversion leads to situations that are potential errors but are not captured. [3] presents a JavaScript program analyzer that can be incorporated into any IDE. The prototype presented helps the developer identify bugs in his code as he writes. However this model is only good for JavaScript, it does not help a developer find bugs when his JavaScript is

coupled with HTML5 and CSS3. The authors agree that JavaScript is used by web developer in tight relation to DOM and their model coupled with a program analysis technique that looks into the HTML file as well can benefit web developers heavily.

Jensen et al. [16] also discusses the dynamic nature of JavaScript and the interconnections between HTML and JavaScript in a webpage. They present a prototype for static analysis of flow of program between the two languages. They represent flow of JavaScript program in the form of flow graphs. These flow graphs helps one understand flow of all possible paths that the code go through and their prototype can point out errors like unreachable code and how JavaScript interacts with the DOM.

In our model we list all possible interconnections in HTML5, JavaScript and CSS which we use to develop a prototype tool that is capable of identifying errors due to missing interconnections. Our model does not generate flow graphs, instead we feel that the developer should know all the possible interconnections between different files and not complying with those dependencies could generate errors or potential threats to the functionality.

2.7 Logical Dependencies

Logical dependencies are dependencies that are identified as the software evolves, these dependencies are not mentioned in the software structure but are implicit [27]. Gustavo A. Oliva et al. [27] classify various categories of logical dependencies. One way of identifying these dependencies is to read the commit history of the repository and

tracking all the files that were changed simultaneously in the same revision. The files that change together very often are likely to have a relation among them [27]. This methodology might be useful for architects who want to track the architecture of the code and make decisions on how to structure it in future throughout the development phase, but this model does not help developers while they write code.

In this chapter we discussed how cohesion and coupling benefit developer to make the code maintainable and reusable. We also discussed why dependency analysis is crucial and how it is done in other languages and we realize that due to the nature of the front end languages there is no direct way to import the existing models into HTML5, JavaScript and CSS3 stack. The researcher community is making an effort in doing static analysis that tackles a part of the problem arising from JavaScript and also indicates that techniques are required that would extend the analysis to other languages of the front end stack: HTML5 and CSS3. In general our work wants to find direct, fine-grained dependencies, while other community solutions to these problems focus on file-level or component/package/module level dependencies. We therefore focus on a model that looks at the details of the entire front end technologies stack and helps developers in decreasing the time frame between bug injection and bug discovery.

CHAPTER 3

SOLUTION OVERVIEW

The different components across the three languages; HTML5, JavaScript, and CSS3 that make a webpage are often tightly coupled to each other. In the development phase other than deploying and running the front end code on a server it is difficult for the developer to find bugs due to unsatisfied dependencies. This problem becomes more difficult to handle as the size of the application grows. A missing dependency could break major functionality of the web page and make it useless for the user or it could create a small cosmetic issue hampering rendering. The developer could benefit from a tool that analyzes his entire code and creates a report of all the missing dependencies.

To tackle this issue we feel that the developer must be aware of all the ways his code can be interconnected. In our model we identify and discuss the entire possible set of interconnections/ dependencies that exist among the three languages. Based on our model, we develop a tool that analyzes the developer's front end code and generates a report of missing dependencies. The goal of this tool is to reduce the time between defect injection and defect discovery. The tool should be accurate in the defect report it generates, therefore in our model we also discuss that some of these dependencies when unsatisfied would not break critical functionality but the developers must be aware of those unsatisfied dependencies and should either eliminate them or write robust code that should be able to handle error situations.

3.1 A Dependency Model for the HTML5, JavaScript and CSS technology Stack

An element of one of the three languages is said to be dependent if it uses another element from the same or a different language of the webpage. Therefore we define dependency as a directed relation between two elements where "A" is said to be dependent on "B" if "A" requires "B" to be present to fulfill a requirement or behavior. We borrow DSM's approach as a starting point to define our model (chapter 2, section 2.2). For the three languages all possible dependencies can be defined in a 3X3 matrix, a total of nine categories of dependencies shown in figure 2. We found that there are dependencies of interest within JavaScript and CSS3 but none within HTML5. Therefore, unlike DSM, there are two cells of the matrix diagonal that cannot be neglected.

From → To	**HTML5**	**JavaScript (JS)**	**CSS3**
HTML5	No dependencies identified.	1. Links from HTML to JS files. 2. Event listeners in HTML file.	1. Class attribute in HTML elements. 2. Other CSS selectors like id, tag name.
JavaScript	Through document object	1. Function calls within a function. 2. global variables.	JavaScript adding CSS class to DOM assuming it exists in one of the CSS files included in the webpage.
CSS3	No dependencies identified.	No dependencies identified.	Dependency to another CSS3 file using an import statement

Figure 2: Dependency matrix of HTML5, JavaScript and CSS3

Within the matrix cells we mention technical ways to create a dependency between two elements. Absence of a dependent element could cause a webpage to be rendered with a slight difference from the expected result or could even cause it to not render at all. Therefore we further define each the dependency to be critical or non-critical. A category of dependency, when unsatisfied, that causes a web page to fail in serving its required function is classified as critical. A category of dependency, when unsatisfied, that causes rendering imperfections in a web page is classified as non-critical.

3.2 Classification of Dependencies

In the following sub sections we discuss each of the above combination.

3.2.1 HTML5 to HTML5

An HTML5 page is information text along set of markup tags. The markup tags help the author organize the text in structured format a browser understands. An HTML element comprises of HTML tags and the text between them. While writing an HTML page a developer should make sure that there is no flaw in the structure of the document. An HTML element can be dependent on another HTML element based on the structure of the HTML code. For example, a `tr` tag is dependent on its enclosing `table` tag or a `submit` button is dependent on its enclosing `form` tag. We are more interested in cross language dependencies and this dependency is well handled by existing tools and IDE's therefore we do not explore it further.

3.2.2 HTML5 to JavaScript

In this type of dependency, an HTML element makes use of a JavaScript element to fulfill a requirement or behavior. JavaScript elements are used in an HTML page to make the web page responsive to user interactions. This category of dependency mostly

includes event listeners and links to external JavaScript files present on the HTML page. An event listener responds to a user's input and an input by the user could be anything from a keystroke to mouse movement on the webpage. For example, an `onclick` event listener attached to a HTML button will invoke the corresponding JavaScript function.

```
<input type="button" onclick="myFunction()">

function myFunction(){
      alert ("A button on the webpage was clicked")
}
```

Code block 3.1: Example of HTML5 to JavaScript dependency

In code block 3.1, when the user clicks on the HTML button, `myFunction` is invoked and an alert is shown to the user. Here we say that the HTML5 component is dependent on the JavaScript function to deliver its complete functionality.

We classify this type as a critical dependency because non-fulfillment of such dependency can cause the web page to fail in serving its functionality, for example, an unavailable `onclick` function attached to submit button could prevent a user from submitting data.

3.2.3 HTML5 to CSS3

The primary use of the class attribute of an HTML element is to style it; therefore we say that the HTML elements are dependent on CSS classes. These classes contain information about how an HTML component should be displayed to the user. There can be more than one CSS class that styles a single HTML element. Absence of an expected CSS class could cause the webpage to render improperly. Two ways how this dependency can be introduced into a webpage is demonstrated in code block 3.2.

```
<div class=”myStyleByClass”></div>
<div id=”myStyleById”></div>

.myStyleByClass
{
       position: relative;
}

#myStyleById
{
      margin: 5px;
}
```

Code block 3.2: Example of HTML5 to CSS3 dependency

Here the two div elements are styled using their id and class attributes. This type of dependency when unsatisfied will not cause the web page to fail in serving its functionality but will cause a cosmetic error, so we classify it as non-critical. It will help the developer to be aware of CSS classes that he would want to define in the CSS file.

3.2.4 JavaScript to HTML5

A JavaScript component and an HTML element can often be tightly coupled in many ways. JavaScript can reference the HTML element by its id, class or tag name and manipulate it; it might change/ remove or add the content inside the DOM element. JavaScript is even capable of binding or changing a JavaScript function in case of a mouse click or a swipe gesture. It is necessary that the JavaScript function finds the DOM element with the id or class it is looking for or else there will be a broken user interface.

```
function changeHTMLcontent(){
      var htmlElement = document.getElementById(“myDivContainer”);
      htmlElement.innerHTML = “Hi, I am a text going to be appended by
                              Javascript to the div”;
}
```

Code block 3.3: Example of JavaScript to HTML5 dependency

For example in the code block 3.3 when you invoke the `changeHTMLContent` function, the contents inside the HTML tag with id `myDivContainer` would be modified to the content assigned to `innerHTML` attribute of the DOM element in the JavaScript. Here we would say that the JavaScript function to execute successfully requires the particular HTML tag with id `myDivContainer` to be present in the HTML document, this makes the JavaScript dependent on the HTML. The JavaScript can also manipulate the styling information of a HTML component. Functionalities like switching tabs within a webpage are implemented using JavaScript. To implement it, the JavaScript changes the display property of multiple div elements on the webpage, example code block 3.4.

```
function changeHTMLdisplay(){
      document.getElementById("myDivContainer").style.display="none";
}
```

Code block 3.4: Example of JavaScript to HTML5 dependency

Here the JavaScript function looks for the HTML element by id `myDivContainer` and changes styling information from a `block` to `none` which makes the JavaScript dependent on the HTML element. We classify this type as a critical dependency because non fulfillment of such dependency can cause the web page to fail in serving its functionality.

3.2.5 JavaScript to JavaScript

Various JavaScript components can be tightly coupled to each other to implement a particular functionality. For example, function calls from one to another or use of global variables as shared object by more than one function. Dependencies within JavaScript are largely defined by the flow of the code. In chapter 2, section 2.3 we discuss the

difficulties in doing flow analysis and static analysis of JavaScript because of its peculiar nature. To organize and maintain code better, developers generally make multiple JavaScript files and place them in logical directories. However, when we run JavaScript code on the browser from an HTML page, the browser requires all supporting JavaScript files to be included in the HTML file, failing to which, the web page will not function as expected. Also, it is essential that all the necessary JavaScript files be included in the HTML page in the required order. Dependency management within JavaScript has been studied extensively by the research community, and discussed in chapter 2 section 2.4. We therefore do not explore this category further.

3.2.6 JavaScript to CSS3

We try to explain this dependency with the help of an example where a JavaScript code changes the class attribute of a DOM element. Here JavaScript makes two assumptions:

1. The CSS class that it is going to add as a class attribute to an HTML tag exists in one of the CSS files included in the webpage.
2. It assumes that the HTML document has the DOM element whose CSS it is trying to change.

```
document.getElementById("myElement").className = "myCssClass";
```

Code block 3.5: Example of JavaScript to CSS3 dependency

Code block 3.5 is an example of this category of dependency. Here you see that first the DOM element `myElement` is referred and then the attribute class is changed to `myCssClass`. Also, executing this line of code would remove the previous value of class attribute. We classify this dependency as critical because the JavaScript interacts with the HTML as well as the CSS and could break functionality if left unsatisfied.

3.2.7 CSS3 to HTML5

A CSS document specifies styling information for a HTML page. One could use either a tag selector or could reference a DOM element using its class, id attribute or tag name. In true sense the CSS document is not dependent on the HTML page; it merely specifies how the HTML page should be presented to the user. We therefore neglect this category of dependency.

3.2.8 CSS3 to JavaScript

CSS elements do not have any relation with JavaScript elements; therefore, we neglect this category of dependency.

3.2.9 CSS3 to CSS3

We could identify only one way for this dependency to occur. Use of an import statement within a CSS file to include another CSS file makes the parent CSS file dependent on the other. Specifying the wrong path to the CSS file to be included can be hard to find because modern developer tools do not point to the location of the CSS file where it was referenced. Therefore, it becomes necessary for the developer to make sure that he includes the CSS file with correct URL; we thus classify it as a critical dependency because if left unsatisfied, it could hamper a large set of functionalities.

In this chapter we listed and described all the possible dependencies that can be found within the three languages; HTML5, JavaScript and CSS3. We also described why some of these dependencies are not of interest to us. Knowledge of these dependency categories will help developers understand the interconnections in their code and also

help them discover and remove defects early in the development cycle. To prove the benefits of our model we realize it into an Eclipse plug-in. This plug-in, HJCDepend, can look into the front end code base (HTML5, JavaScript and CSS3) and generate a report listing unsatisfied dependencies if any exists. Through this tool our goal is to give the developer an efficient mechanism that could help him identify bugs as soon as he injects them. Our model does not define how the developer should fix the defects but it gives the developer a good understanding of the nature of the defect. Using such a tool, developers would be able to code more confidently and would deliver better quality code by increasing the overall defect detection percentage. We discuss the technical implementation of our model in next chapter and we validate the model by conducting a research study on a group of developers in chapter 5.

CHAPTER 4

HJCDepend IMPLEMENTATION

To realize our model we built a tool, HJCDepend, that could parse, analyze and report dependency related errors in HTML, JavaScript and CSS files. The huge user base of Eclipse for developing web applications [21, 22] and we leverage on the fact that a large community of developers are comfortable using the development environment Eclipse has to offer. This led us to the decision to realize HJCDepend as an eclipse plug-in. It is designed to serve the developer as he writes code without having the need to write special code like a configuration file to run it. HJCDepend's primary role is to identify components that are dependent to other components in the code in some other file. Each line of code is analyzed to collect all the required data that would help in generating a comprehensive list of unsatisfied dependencies. While collecting data we keep track of its filename and line number so that the developer will know where to find the defect. The dependencies that fail to be fulfilled are reported as errors or warning (decided by the criticality of the dependencies we discussed in our previous chapter) to the developers.

On a browser an HTML file is an entry point to the entire front end code. This plug-in also takes in the path of the HTML file you want to analyze as an input and returns a list of errors and possible warnings. For convenience this plug-in does not require you to specify the path of the HTML anywhere, you could open the HTML file you want to analyze and then hit the "Run HJCDepend" button from the plug-in menu. For achieving our final goal of realizing the tool into an Eclipse plug-in, we started the

development by writing three independent modules in Java that could parse HTML5, JavaScript and CSS3. In section 4.1 we define what we want to achieve from the particular module and then we dive into the implementation to complete the requirement. In sections 4.1.1, 4.1.2 and 4.1.3 we discuss how we parse the three languages. We then talk about comparing the data we collect during the parsing process and how we implemented the entire functionality into an Eclipse plug-in in sections 4.2 and 4.3. We conclude this section by giving an overview of installing and running HJCDepend.

4.1 Parsers

Our primary requirement for all the parsers was that they should output as much data as possible in an hierarchical format or a tree structure for example an Abstract Syntax Tree. Also, a few other key features that we wanted from our parsers were:

- Provide a convenient and elaborate API to access the output data.
- Be robust in the face of errors and report syntactic errors too.
- Report the filename and line number of where the data or tokens were found wherever required.
- Since we have to only read files to analyze them, therefore the parser we choose does not need to have functionality to write back to the files. However it should be capable of outputting the input in a tidy format.

4.1.1 Parsing HTML5

Considering our above requirements for a parser we found that Jsoup[4] is a good option for parsing HTML5. Jsoup is a java library for parsing HTML files with support to latest HTML5 tags, it implements WHATWG HTML5[5] specifications. It parses the HTML into a DOM similar to that in a browser. It provides an API that makes accessing

[4] http://jsoup.org/
[5] http://www.whatwg.org/specs/web-apps/current-work/

data of the DOM convenient. It has the ability to read HTML files from disk or it could also fetch the HTML document from a URL and parse it. One drawback of this parser is that it does not maintain filename and line number. To retain this information we built an extra layer in our module that takes care of this requirement.

In our module we read the HTML5 file from the disk and then we parse it using Jsoup. Jsoup then returns a Java object of type “Document”. A “Document” object consists of a list of HTML elements the parser found in the HTML5 file. The “Element” class can give information about the tag name and attributes of any particular HTML5 tag. The “Document” class is a child for class “Element” and the base abstract class is “Node”.

After parsing the HTML file, we iterate over all the elements found and extract the following information:

- Value of all attributes named “class”, this would help us find HTML5 to CSS3 dependencies.
- All function names that are attached as event listeners to any element; this would help us find HTML5 to JavaScript dependencies.
- The parameter count of each function attached as the event listener, this would help us report any warnings related to function variadicity.
- The JavaScript and CSS3 file links using the script and link tags.

We then try to read all the JavaScript and CSS3 files mentioned inside the HTML file, we report unavailable files. Our implementation is capable of reading the JavaScript and CSS3 files either from the disk or from a URL. We store the above data along with

information about where we found the data i.e., the filename and line number in Hash Maps for comparing the results and generating output discussed in section 4.2.

4.1.2 Parsing JavaScript

Rhino[6] is a JavaScript parser written by the Mozilla Foundation in Java. It fulfills our requirement of presenting the parsed output as an Abstract Syntax Tree (AST) and provides an API to interface with the tree. It maintains the filename and line number of each token making it easy for us to pinpoint where we found particular token or data. Before we create the AST, we first check each JavaScript file for any syntactic errors and report if found in any of the included JavaScript files. Once we are confident that we can parse each file, we then combine them into one file, which Rhino parses. We do this because all JavaScript that is included in a web page is combined by the browser and executed as a single file. When we combine the files, we make sure that they are inserted into the single JavaScript file in the same order as they appear on the HTML page.

We need to extract the following information about JavaScript files being analyzed:

- Name and parameter count of all functions declared in the JavaScript, we would use this data to verify that the functions bound to HTML5 elements are all present in the included JavaScript files.
- All calls made to methods that can fetch and write onto the DOM element. We want this data to make sure that any IDs or any other selectors the JavaScript uses to reference the DOM are present in the HTML file. From these methods we thus generate a list of all selector elements.
- All the JavaScript variables that reference to unavailable DOM element that could generate errors if any property inside them is tried to be accessed or modified.

Figure 3 is an example of an AST the parser generates when we give Rhino the JavaScript string: `document.getElementById("someHTMLId");`

[6] https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino

```
{
    "type": "Program",
    "body": [
        {
            "type": "ExpressionStatement",
            "expression": {
                "type": "CallExpression",
                "callee": {
                    "type": "MemberExpression",
                    "computed": false,
                    "object": {
                        "type": "Identifier",
                        "name": "document"
                    },
                    "property": {
                        "type": "Identifier",
                        "name": "getElementById"
                    }
                },
                "arguments": [
                    {
                        "type": "Literal",
                        "value": "someHTMLId",
                        "raw": "'someHTMLId'"
                    }
                ]
            }
        }
    ]
}
```


Figure 3 Example of Abstract Syntax Tree

We traverse the complete tree generated by Rhino and search for specific tokens for example the ones that represent function call or access to JavaScript variables. We look for specific functions that are capable of referencing HTML elements (see list in APPENDIX A); we create a list of all such references and store it. Similarly, we calculate the number of parameters a function expects. This data and information about the location of individual tokens is stored in Hash Maps.

4.1.3 Parsing CSS3

We defined in our previous chapter that we do not identify any dependency arising from CSS3 file to HTML5 or JavaScript. However, we define a specific case when a CSS can be dependent upon another CSS file by using the "import" statement. The parser thus selected for this task is phloc-css[7]. Phloc-css is written in Java and can parse CSS3 fed to it in form of files or string. This parser provides an elaborate and convenient API to access the parsed output. It provides functions that return all the selectors in the CSS file and functions that return a filtered list of selectors. It is robust in case it is fed with CSS3 that has syntactic errors and outputs a detailed report in face of syntactic errors.

Using this parser like JavaScript we first verify that all CSS3 files referenced by the HTML page are parse able, that is, they do not have any syntactic errors, we report errors if found. We then get all the locations of the other CSS files that are made using imports statements. Import statements can reference another CSS files using a relative path (same domain) or an absolute URL to another domain. We fetch these files, wherever they are located. If the tool is unable to read or fetch the specified file, a CSS to CSS dependency is reported.

4.2 Binding the parsers to compare their output

Once we collect all the required information from the parsers, we then run an analysis on that data to determine unsatisfied dependencies defined by our model. To bind all the parsers and compare their output we create a class that acts as the entry point

[7] http://www.phloc.com/view/p-1018/Open-Source/

for HJCDepend core functionality. This class interfaces with the various parsers we defined in the above sections and then runs various comparisons after getting output from each one of them. We also maintain a singleton object that stores all the data structures used by various parsers, none of the parser classes store any data within itself. For generating output report we do the following comparisons:

- We compare all the event listener functions mentioned in the HTML5 document with the list of functions that are present in the included JavaScript files. In case we do not find a function in JavaScript that was mentioned in the HTML document, we report unsatisfied HTML5 to JavaScript dependency.
- We compare the list of class attribute values with the all the CSS selectors found in the CSS files. We report a warning if a class is not defined in the CSS files, this is a HTML5 to CSS3 unsatisfied dependency.
- We compare the list of DOM ids and classes present in the HTML document with the list of HTML references made by the JavaScript. We report a warning if the reference to unavailable HTML element is not being written or else it this is reported as error if a property is tried to be accessed on that element.

Once we have all the unsatisfied dependency we ask each parser to report their exact location in the code they analyzed. Thus the output of this binder is a list of unsatisfied dependencies with a small description about the error or warning which helps the developer understands its nature.

4.3 Embedding the core functionality into Eclipse

Eclipse runs subproject Plug-in Development Environment (PDE) that provides developers with a convenient development environment to develop plug-ins in Java for the IDE. To implement our core functionality, we create a class that extends “ViewPart”. Eclipse requires us to extend this class to create a view inside the IDE. In our plug-in we fetch the path of the currently opened HTML file in the Eclipse editor for reasons defined in the beginning of this chapter. We feed this path to our binder class which returns us a

report of unsatisfied dependencies. This report is shown to the user in a tabular format. Each row of this table consists of:

- The severity of the unsatisfied dependency
- The dependency type defined by our model
- A small description of what the defect is and why it might have been caused.
- The filename and line number of where the defect was found in the code.

For the convenience of the developer analyzing large codebase, we take the developer to the specific line number where the defect was found upon double clicking a defect row present inside the report.

4.4 Installation of HJCDepend

To install HJCDepend a developer should place the Jar file inside the plugin directory of his Eclipse installation. HJCDepend does not require any other libraries to be included while installation, we pack the entire required parser library within our Jar file. HJCDepend can be accessed like any other plug-in in Eclipse using the "Windows" option in the menu bar. To run the analysis, the user would open the HTML5 file in the editor and then press the "Run HJCDepend". The user would be then presented the list of defects found in the source code if present any.

CHAPTER 5

VALIDATION

We list and define all dependencies between HTML5, JavaScript and CSS3 in chapter 3. We built an Eclipse plug-in, HJCDepend, (discussed in chapter 4) that can identify these dependencies in a body of code and can report dependency related errors. One of the goals of defining this model and implementing it into a plug-in is to assist the developer in discovering and removing defects early in the development cycle. In this chapter we discuss a research study conducted on a group of developers to measure the productivity and efficiency of developers using HJCDepend in discovering and removing defects by calculating the following metrics:

- *Precision* - The fraction of retrieved defects that are relevant, in other words, the percentage of defects found by a developer that were actually defects out of the total defects found by him. This tells us the accuracy of the developer in finding defects.
- *Recall* – The fraction of relevant defects found by a developer out of the total defects that are present in a body of code. This would tell us what percentage of the total defects present in the body of code was found by the developer.
- *Defect discovery rate* – The average time taken by a developer to find a defect in a body of code. This will indicate how fast a developer finds defects.
- *Defect removal rate* – The average time taken by a developer to remove/ fix a defect in a body of code. This will indicate how efficient a developer is in removing defects.

We describe the protocol and our methodology to calculate these metrics in sections 5.1 and 5.2. We then present the results of this research study in section 5.3 and discuss the conclusion in section 5.4.

5.1 Protocol

We conducted a research study on a group of developers who were asked to debug an existing body of code for dependency related defects with the help of HJCDepend and another group of roughly equivalent skill set of developers that debugged the same code but without HJCDepend. Each developer was given a total of 75 minute to do two tasks. First, the two groups of developers were not allowed to make any changes to the code and were asked to report as many defects as they could find in the source code. They were not aware of the total number of actual defects present in the code and were told that once they were convinced that they have found all the defects in the source code, in the remaining time of the study their second task was to remove the defects and report each defect they fixed.

5.1.1 Environment for Participants

Each participant was given a Windows 7 desktop with an almost similar configuration. One of the two groups, group A, was allowed to use any text editor or IDE they wanted to. They used editors and IDEs like Notepad++, Word Pad and Dreamweaver. Group B was given an installation of Eclipse along with HJCDepend. The two groups were given identical source code (see APPENDIX C) that had been seeded with dependency related defects that HJCDepend could detect.

5.1.2 The Source Code

The source code used for this research study was a two-page web application which consisted of 2 HTML files (total 100 lines), 4 JavaScript files (total 85 lines) and 3 CSS files (total 142 lines). It was seeded with dependency related defects. Figure 2 is a

matrix of possible dependencies we discussed in chapter 3. In figure 4 we present a similar matrix but with the total number of defects that were injected in the source code according to the dependency type. JavaScript to CSS3 dependency defect is outside the scope of HJCDepend and other three cells of the matrix that have value "none" are the dependency categories which we ignore in our model (chapter 3, section 3.2.1, 3.2.8 and 3.2.9). The defect density for HTML, JavaScript and CSS in the source code was 12%, 8% and 0.7% respectively. See APPENDIX C.1 – C.9 for complete source code and C.10 for the list of defects with description, filename and line number.

To -> From	HTML5	JavaScript	CSS3
HTML5	none	9	3
JavaScript	6	1	none
CSS3	none	none	1

Figure 4 Summary of defect types in infected source code

In an effort to get normalized results, the source code was also seeded with one bug that was not within the scope of HJCDepend to find (JavaScript to JavaScript dependency type). The source code used for this research study was kept confidential from the principal investigator until fifteen minutes before the research study. This was done to remove any kind of bias.

5.1.3 Participants

The identity of the participants was kept anonymous during this entire study. We conducted a survey to know the skill set and experience of the participants in HTML5, JavaScript and CSS3. In this survey we asked the participants about their past experience and knowledge in HTML5, JavaScript, CSS3 and Eclipse IDE (see APPENDIX B.1 for complete list of questions asked in this survey). A total of 27 participants registered for

the study and 24 showed up. We divided the participants into two groups of roughly equivalent skill sets based on the results of this survey.

5.2 Empirical Study

The first task of the two groups was to identify and report all the defects they could find inside the given source code without making any changes to it. They were asked to report their group number, the time they took to find the defect, a description of the defect, filename and line number where they found the defect using a Google survey form (see list of questions asked to report the defect in APPENDIX B.2). All participants were provided online stopwatches and were told to start it when they began to look for a defect and stop it when they found a defect. Analyzing all the defects reported by each developer, we calculate the precision for the two groups. We did not tell the developers the total number of defects present in the source code and asked the developers to report as many defects as they could find. Therefore, the recall was calculated as the fraction of relevant defects found by a developer out of the total defects that were present in the body of code. To calculate the defect discovery rate for each group, we asked the developers to report the time they took to find each defect.

In the remaining time of the study, the two groups were asked to remove each defect they had found in the previous task. The goal of this task was to measure the average time taken by participants in each group with and without HJCDepend to remove each defect. They were also asked to log their group number, the time it took for them to remove each bug, a short description of the steps they took to remove the bug and the

name and line number of the files they edited to remove the bug through a google survey form (see list of questions to report a fix in Appendix B.3). Using this information we calculate the defect removal rate of the two groups.

We then asked both the groups to take an exit survey to share their experience of the research study and we also asked group B to report their experience of using HJCDepend. The questions in exit survey for group A and group B can be found in APPENDIX B.4 and APPENDIX B.5 respectively.

5.3 Results and Observations

We discuss the results and observations of the two tasks and exit survey in the following subsections. 22 valid unique participant's responses were received from a total of 24 participants. One unique participants in each group did not report any defect discovery or removal and did not take the exit survey. Thus they are discarded from the study's results (both groups had 11 unique participants).

5.3.1 Defect Discovery - Results and Observations

For this task a total of 157 defects were reported by the two groups. The participant were told to submit one Google survey form for each defect they discovered however there were 11 responses which had two defects each. Thus, the actual number of defects discovered by the two groups was 168. Out of the 168 reported defects, the group without HJCDepend (group A) reported a total of 56 defects in which 45 defects

were valid and the group with HJCDepend (group B) reported a total of 112 defects in which 110 defects were valid.

Therefore, the precision of finding defects for group A was 45/56 (80.35%) and for group B it was 110/112 (98.21%). There were a total of 20 defects in the source code given to the participants. The total defects that could be discovered by each group was 220 (11 participant per group multiplied by total defects in the source code, i.e., 20). Therefore the recall for group A was 45/220 (20.45%) and recall for group B was 110/220 (50.00%). (See APPENDIX C.14 for a detailed report of defects reported by each participant).

Figure 5 is a graph representing the number of valid defects reported by members of both groups on basis of bug id and dependency type defined in chapter 3. The x-axis represents the bug Id and the y-axis represents the number of bugs reported. See APPENDIX C.10 and C.11 for description of bug id and the data source of graph in figure 5, representing number of responses by both groups respectively.

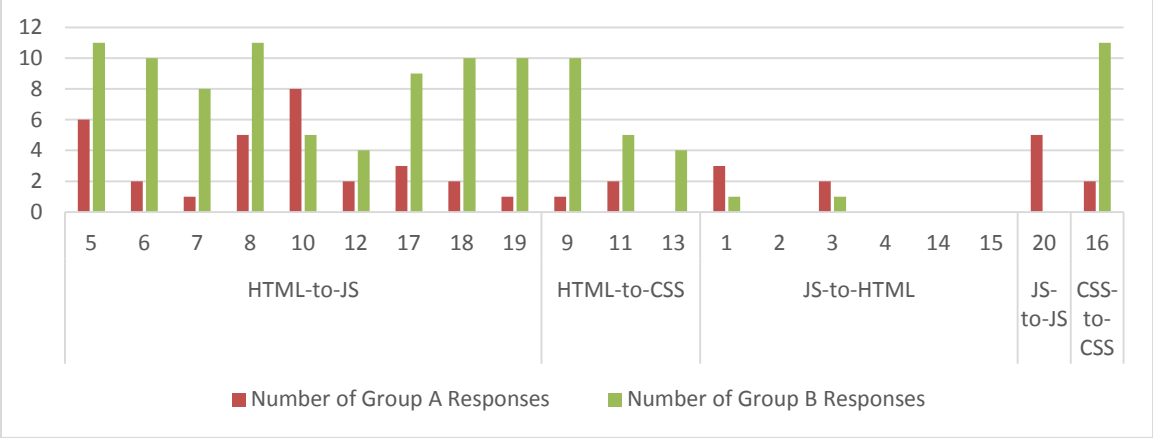

Figure 5 Summary of valid defects reported by both groups

We observe that defects 2 and 4 were not reported by either of the two groups. We believe this is due to the fact that these defects occurred as a by-product of defects 1 and 3 respectively. Also, defects 1 and 3 were reported by a low number of participants because it is difficult to detect them with or without HJCDepend unless defect 17 is fixed. The two defects 14 and 15 were also not reported by any of the groups. Unlike defects 2 and 4, defect 14 was not a by-product of other defects; however they both could not be detected by a developer or HJCDepend unless defect 10 is fixed. Defect 13, was reported only by group B and was not a critical defect defined according to our model. HJCDepend shows this defect as a warning to the developer and it is for the developer to decide whether he should resolve it or not. Defect 20 was the one defect that was not within the scope of HJCDepend and was a by-product of defect 10, it was reported by group A only. Perhaps group B understood the fact that it is a by-product of defect 10 and decided not to report it. The overall results signify that the number of bugs reported by group B was much higher than that identified by group A. Overall in figure 5 we observe that both groups combined reported a very low number of JavaScript to HTML related defects, it is because these defects cannot be detected by HJCDepend or a developer before we resolve defects related to incorrect file paths to JavaScript from the HTML page.

Figure 6 is a graph representing the average time taken by both groups to report valid defects on basis of bug id and dependency type defined in chapter 3. The x-axis represents the bug Id and the y-axis represents the average time taken by participants in both groups to report each bug. See APPENDIX C.10 and C.15 for description of bug id

and the data source of graph in figure 6, representing average time taken by both groups' participants respectively.

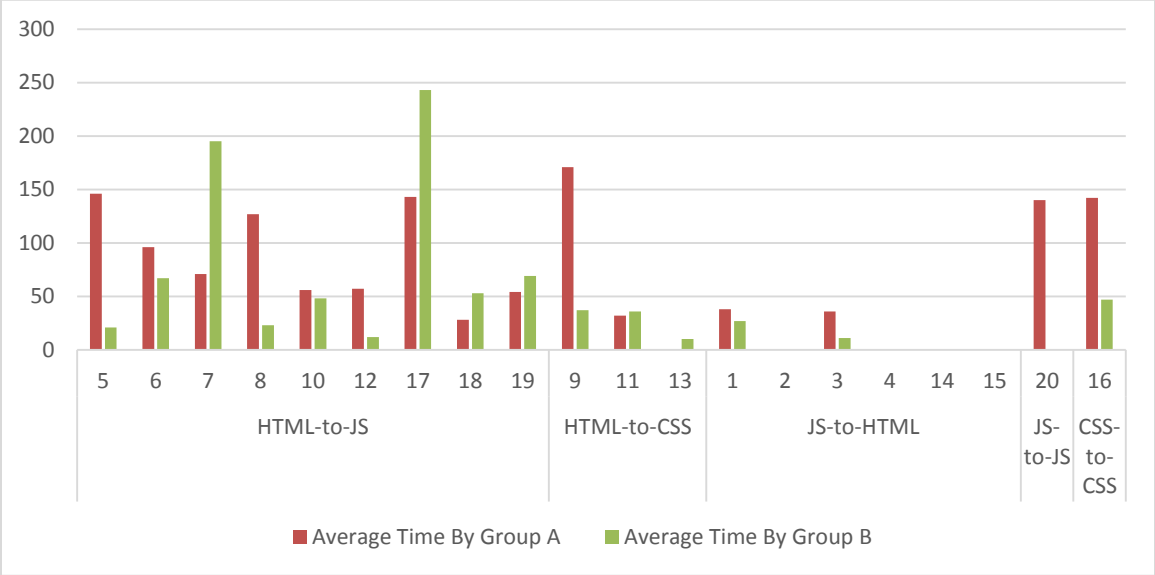

Figure 6: Summary of average time taken to report defects by both groups

Group A took lower reporting time for defects 7, 11, 17 and 19. For bugs 7 and 17, there was one response each that reported a high time taken to find these bugs driving the entire average very high. Whereas the response time of defects 10, 11 and 19 is approximately similar for both the groups.

For group A the total time taken to report 45 valid defects was 4321 seconds, their average time to discover one defect is 4321/45 (96.02 seconds) and for group B the total time taken to report 110 valid defects was 7628 seconds, their average time to discover one defect is 7628/110 (69.34 seconds).

5.3.2 Defect Removal Results - Results and Observations

For this task a total of 135 defects were reported by the two groups. The participant were told to submit one Google survey form for each defect they removed however there were 13 responses which had two defects each. Thus, the actual number of defects discovered by the two groups was 148 out of which 13 defect fixes reported were not required. See APPENDIX C.12 summary of number of bugs fixed and the average time taken by each group to fix each bug. Out of the 148 reported defects, the group without HJCDepend (group A) reported a total of 55 defects in which 43 defects were valid and the group with HJCDepend (group B) reported a total of 93 defects in which 92 defects were valid.

For group A the total time taken to report 43 valid defects was 2129 seconds, their average time to remove one defect was 2129/43 (49.51 seconds) and for group B the total time taken to remove 92 valid defects was 3404 seconds, their average time to remove one defect was 3404/92 (37 seconds). This data tells us that the average time for defect removal of group B was lower than group A. Therefore, group B had a better defect removal rate.

5.3.3 Exit Survey Results

None of the participants found the complexity of the given code very hard. This tells us that the participants in both groups could understand the coding task within the time constraints given, so we could safely say that we do not have skewed results based on lack of code understanding. Figure 7 represents the responses of the two groups regarding the complexity of the code.

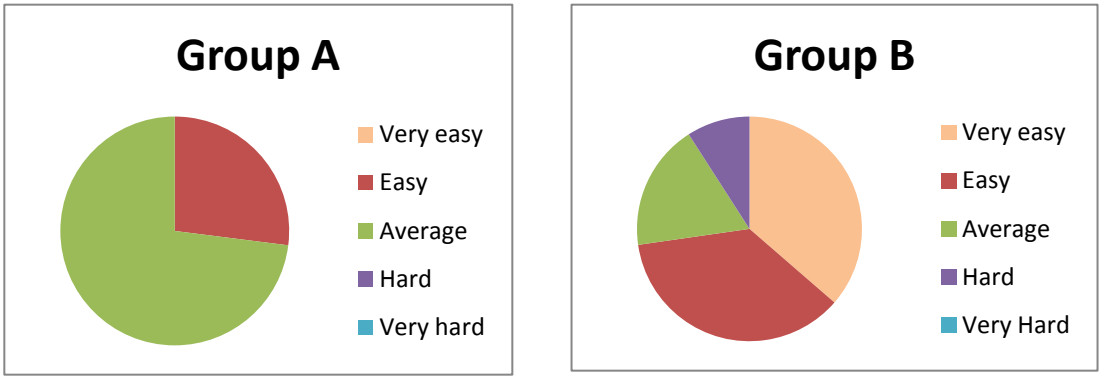

Figure 7: Complexity of infected source code rated by participants

We asked group B a few questions on their experience of using HJCDepend to find and fix bugs. We present their responses in figures 8 and 9.

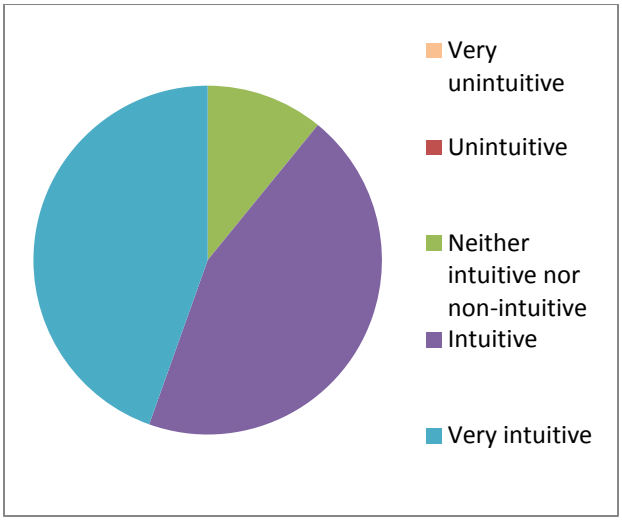

Figure 8: Rating of intuitiveness of HJCDepend by participants

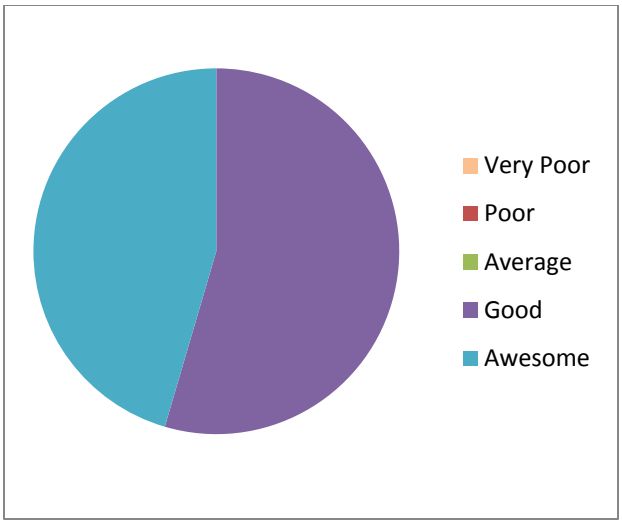

Figure 9: Rating of experience using HJCDepend by participants

The responses of the post survey result indicate that HJCDepend is intuitive and helps developers to find defects. Some participants appreciated the concept of the tool and suggested us to add features that identify defects other than dependency related like "running JSLint in background".

5.4 Conclusion

In this chapter we presented the results of a research study to validate the underlying model (defined in chapter 3) of HJCDepend. We discussed the number of defects found and removed by each group and the time they took to perform these tasks. With this information we derived the precision, recall, defect discovery rate and defect removal rate.

From the results discussed in the previous section we conclude that using HJCDepend developers find more number of bugs when compared to developers trying to find defects using conventional techniques. A t-test results show that the difference of

defects discovered between the 2 groups is significant ($p = .002$ and $p = .02$ for group A and group B respectively). The precision and recall for group B is significantly higher when compared to group A.

The lower time taken to find and remove each defect indicates that group B has a better defect discovery and removal rate. We can thus conclude that HJCDepend and consequently our underlying model helps the developers discover and remove a higher number of relevant defects efficiently when compared to developers using conventional techniques.

CHAPTER 6

SUMMARY AND FUTURE WORK

6.1 Summary of the Present Work

Web developers are making extensive use of HTML5, JavaScript and CSS3 to develop web pages. Even a mid-sized application's code can get difficult for the developer to find and fix defects. Understanding dependencies between these three languages can help developers discover bugs early in the development cycle. Dependencies are created when a component in the front end code requires another component from another source to complete its functionality. The tight coupling of components within these languages makes the code complex and hard to debug for defects.

In chapter 2, we talked about community solutions to managing defects and the extensive research in JavaScript static and flow analysis. We found that there are multiple solutions to solve dependency management problems in JavaScript but there has been very little effort to define and identify the extent of the problem in detail. Existing models like DSM cannot be directly applied to this language stack due to the nature of these languages. JavaScript's peculiar features like eval, make this language difficult to debug for errors. A large group of researchers are looking into static analysis of JavaScript which help them find security related defects in the code. Researchers also point out that JavaScript is used by web developers tightly with DOM and a model that looks into the HTML file, JavaScript and CSS file can benefit web developers.

We defined a model which categorizes all the dependencies within the three languages. This model helps the developer find bugs faster and easily. Understanding the nature of the bug, derived from our model, increases the efficiency of developer in fixing it without injecting more. We stated why some dependencies when unsatisfied can break critical functionality and on the other hand some of them are only capable of creating minor issues.

We realize our model into an Eclipse plug-in, HJCDepend, which reads the developer's front end code and analyzes it for dependencies defined by our model. HJCDepend generates a report of defects it finds for the developer. The plug-in does not make any suggestions on fixing the issues, it fulfills our goal in finding the injected bugs as early as possible in the development cycle.

We validate our model and the tool by conducting a research study. Two groups of developers try to find defects in an infected source code. One group uses HJC Depend while the other uses conventional techniques to identify defects. We observe that the group using HJCDepend designed based on our model performs better by having a higher precision and recall for the defects in a source code when compared to a group of developers using conventional techniques to debug. The study also demonstrates that the overall defect discovery and removal rate is enhanced using HJCDepend.

6.2 Future Work

HJCDepend is a prototype implementation of the model we defined in chapter 3. Through this implementation we proved how useful this model can be to developers in finding and removing defects, however, the uses of this model is not limited to finding defects. This model can be implemented to serve more purposes.

A deployment or operations team will benefit from an implementation of this model that, ulike HJCDepend, does an analysis of all HTML files in a web application and lists all unsatisfied dependencies in the code. The deployment team can run this implementation before every code deployment to make sure they are not missing any files.

An architect can benefit from an implementation of this model which lists all the dependencies present in a body of code. It would help him understand the overall modularity of the code by knowing how and where the three languages of the source code files are dependent upon each other. With this information they can make better decisions on where code related to new functionalities should be placed. They can design more modular architecture for future projects by studying their code for past projects using this model.

Web applications can have front end code that spans over more than hundred files, it becomes cumbersome for new developers joining the team to understand the code and how source files are related to each other. They would benefit from an

implementation of this model that visualizes the dependencies in a code. This visualization would consist of connections between the source files. Each source file could be represented as a node and a node can have multiple connecting lines to other nodes, these connections will be based on the dependencies defined in chapter 3. Clicking on a connection inside the visualization should bring forward details like, the type of dependency, the files required to satisfy the dependencies and how the dependency was created. This will help them in understanding web page functionality implementation faster and better.

REFERENCES

[1] Browning, T.R., "Applying the design structure matrix to system decomposition and integration problems: a review and new directions", 2001, Engineering Management, IEEE Transactions.

[2] Book: Eppinger, S.D., and Browning, T.R., "Design Structure Matrix Methods and Applications".

[3] Jensen, S.H., Møller, A., and Thiemann,P., "Type Analysis for JavaScript". SAS '09 Proceedings of the 16th International Symposium on Static Analysis, Springer-Verlag Berlin, Heidelberg.

[4] Sangal N., Jordan E., Sinha V., Jackson, D., "Using Dependency Models to Manage Complex Software Architecture". OOPSLA '05 Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, ACM New York, USA.

[5] Chugh R., Meister, J.A., Jhala, R., Lerner, S., "Staged Information Flow for JavaScript" Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation, Pages 50-62.

[6] Wei, S., and Ryder, B.G., "Practical blended taint analysis for JavaScript". Proceedings of the 2013 International Symposium on Software Testing and Analysis, ACM New York, NY, USA.

[7] Madsen, M., Livshits, B., Fanning, M., "Practical Static Analysis of JavaScript Applications in the Presence of Frameworks and Libraries". ESEC/FSE 2013 Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, Pages 499-509, ACM New York, NY, USA

[8] Meyers, T.M., Binkley, D., "An empirical study of slice-based cohesion and coupling metrics", ACM Transactions on Software Engineering and Methodology (TOSEM).

[9] Gui, G., Scot, P.D., "Coupling and cohesion measures for evaluation of component reusability" Proceedings of the 2006 international workshop on Mining software repositories. Pages 18 - 2, ACM New York, NY, USA.

[10] Counsell, S., Swift, S., Tucker, A., "Object-oriented Cohesion Subjectivity amongst Experienced and Novice Developers: an Empirical Study", Volume 31 Issue 5, September 2006, ACM New York, NY, USA.

[11] Rizvi, S.W.A., and Khan, R.A., "Maintainability Estimation Model for Object-Oriented Software in Design Phase (MEMOOD)", JOURNAL OF COMPUTING, VOLUME 2, ISSUE 4, APRIL 2010, ISSN 2151-9617.

[12] Alagar, V.S., Li*. Q., Ormandjieva, O.S., "Assessment of Maintainability in Object-Oriented Software". Proc. 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39), 29 July -03 Aug. 2001, vol. 19, pp. 194 – 204, Santa Barbara, California, USA, 2001.

[13] JDepend: http://clarkware.com/software/JDepend.html

[14] Book: Greg, F., "Instant Dependency Management with RequireJS How-to".

[15] Book: Sugrue, J., "Beginning Backbone.js".

[16] Jensen, S.H., Møller, A., Madsen, M., "Modeling the HTML DOM and Browser API in Static Analysis of JavaScript Web Applications". Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, ACM New York, NY, USA.

[17] Obviel: http://www.obviel.org/

[18] BackBone.JS: http://backbonejs.org/

[19] Richards, G., Lebresne, S., Burg, B., and Vitek, J.. "An analysis of the dynamic behavior of JavaScript programs". In Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation, PLDI '10, pages 1-12. ACM, 2010.

[20] Amalfitano, D., Fasolino, A.R., Polcaro, A., Tramontana, P., “Comprehending Ajax Web Applications by the DynaRIA Tool”. Published in Program Comprehension (ICPC), 2010 IEEE 18th International Conference on June 30 2010-July 2 2010.

[21] Geer, D., “Eclipse Becomes the Dominant Java IDE”. IEEE Computer Society.2005.7

[22] Vaughan-Nichols, S.J., “The battle over the universal Java IDE”. Published in Computer (Volume: 36, Issue: 4), Sponsored by IEEE Computer Society”.

[23] Introduction to RequireJS
http://javascriptplayground.com/blog/2012/07/requirejs-amd-tutorial-introduction/

[24] Martel.M and S. Martel.S.. (2002). "Extreme Programming: Rapid Development for Web-Based Applications". IEEE Internet Computing, 6(1) pp 86-91 January/February 2002.

[25] Stephens, M., Rosenberg, D. (2003). "Extreme Programming Refactored: The Case Against XP". Apress, 2003.

[26] Nikiforakis, N., Invernizzi, L., Kapravelos, A., Acker, S. V., Joosen, W., Kruegel, C., Piessens, F., and Vigna, G. “You are what you include: large-scale evaluation of remote javascript inclusions”. In Proceedings of the 2012 ACM conference on Computer and communications security, CCS ’12, pages 736–747, Raleigh, NC, USA, 2012. ACM.

[27] Oliva, G., Santana, F., Gerosa, M. A., and de Souza, C. R. B., “Towards a Classification of Logical Dependencies Origins: A Case Study,” In Proceedings of the Joint ERCIM Workshop on Software Evolution and International Workshop on Principles of Software Evolution (IWPSE-EVOL 2011), Szeged, Hungary.

APPENDIX A

IMPLEMENTATIONAL DETAILS OF HJCDEPEND

A.1. List of event Listener functions searched by HJCDepend for HTML to JavaScript dependencies inside HTML file

- onclick
- ondblclick
- onmousedown
- onmousemove
- onmouseover
- onmouseout
- onmouseup
- onkeydown
- onkeypress
- onkeyup
- onblur
- onchange
- onfocus
- onreset
- onselect
- onsubmit
- onunload
- onload
- onerror
- onscroll
- onresize

A.2. List of JavaScript inbuilt functions ignored by HJCDepend in HTML file

- alert
- decodeURI
- decodeURIComponent
- encodeURI
- encodeURIComponent
- escape
- eval
- isFinite
- isNaN
- Number
- parseFloat
- parseInt
- String
- unescape
- atob
- blur

- btoa
- clearInterval
- clearTimeout
- close
- confirm
- createPopup
- focus
- moveBy
- moveTo
- open
- print
- prompt
- resizeBy
- resizeTo
- scroll
- scrollBy
- scrollTo
- setInterval
- setTimeout
- stop
- console.log

APPENDIX B

SURVEY QUESTIONNAIRE

B.1. Pre Participation Questionnaire

- Rate yourself on your knowledge of Eclipse.
- Rate yourself on your knowledge of HTML5.
- Rate yourself on your knowledge of JavaScript.
- What is your past experience in HTML5?
- What is your past experience in JavaScript?
- What is your past experience in CSS3?
- What is your GPA?
- What time would you like to take the coding exercise?

B.2. Defect Reporting Questionnaire

- Please enter your unique identifier
- Group A or B?
- Please describe the bug in one or two lines.
- Please enter the time (mm:ss) from the stop watch it took you to find this bug. (minutes:seconds)
- Please enter the file name(s) where you found the bug.
- Please enter the Line Number(s) where the bug is located.

B.3. Defect Fixing Questionnaire

- Please enter your unique identifier
- Group A or B?
- Please describe how you fixed the bug in one or two lines.
- Please enter the time (from the stop watch) it took you to fix this bug.
- Please enter the filename(s) where you fixed the bug.
- Please enter the Line Number(s) which changed to fix the bug.

B.4. Post Participation Questionnaire (Group A: not using HJCDepend)

- Rate the complexity of the given code.
- Rate your experience in finding bugs in the given code.
- Rate your experience in fixing bugs in the given code.
- Other Comments?

B.5. Post Participation Questionnaire (Group B: using HJCDepend)

- Rate the complexity of the given code.
- Rate your experience of using Eclipse outside of using HJCDepend as an IDE for developing code in languages you just used.
- Rate the intuitiveness of HJCDepend.
- Rate your experience of using HJCDepend for finding bugs in the code.
- Would you use or recommend HJCDepend to other people in future?
- Suggestions? What more do you think this tool should be able to do?

APPENDIX C

INFECTED SOURCE CODE GIVEN TO THE PARTICIPANT FOR RESEARCH STUDY

C.1. index.html

```
Line 1: <!DOCTYPE HTML>
Line 2: <html>
Line 3: <head>
Line 4: <title>Ajax Basics</title>
Line 5: <link rel="stylesheet" href="./css/styles1.css" type="text/css"
/>
Line 6:
Line 7:
Line 8:
Line 9: </head>
Line 10:
Line 11: <body>
Line 12:    <header> Understanding Ajax </header>
Line 13:
Line 14:
Line 15:    <div class="mainContainer">
Line 16:          <div>
Line 17:
Line 18:                <div id="pane1-btn" class="tab-button"
onclick="togglexTab('pane1')">Ajax Basics: Part I</div>
Line 19:                <div id="pane2-btn" class="tab-button"
  onclick="toggleTab('pane2','pane2')" style="background : #5F789E"
>Ajax Basics: Part II</div>
Line 20:
Line 21:          </div>
Line 22:          <div class="tab-pane" id="pane1">
Line 23:
Line 24:                <div class="form-container">
Line 25:
Line 26:                      <h3>Data from JSP, Result Shown in Box
Below</h3>
Line 27:
Line 28:                      <!-- Content will get appended to this
div by getDataFromJsp -->
Line 29:                      <div class="result"
id="jspResultContainer"></div>
Line 30:                      <input type="button" value="Get Data From
JSP"
Line 31:                            onclick="getDataFromJsp()" />
Line 32:
Line 33:
Line 34:                </div>
Line 35:
Line 36:                <div class="form-container">
Line 37:
Line 38:                      <h3>Data from Servlet, Result Shown in
Box Below</h3>
Line 39:                      <!-- Content will get appended to this
div by getDataFromServlet -->
Line 40:                      <div class="result-box"
id="servletResultContainer"></div>
Line 41:                      <input type="button" value="Get Data From
Servlet"
```

```
Line 42:                              onclick="getDataFromServlet('show-
time')" />
Line 43:
Line 44:
Line 45:                </div>
Line 46:                <div class="form-container">
Line 47:
Line 48:                      <h3>Data from RestAPI, Result Shown in
Box Below</h3>
Line 49:                      <!-- Content will get appended to this
div by getDataFromRest -->
Line 50:                      <div class="result-box"
id="restResultContainer"></div>
Line 51:                      <input type="button" value="Get Data From
REST API"
Line 52:                            onclick="getDataFromRest('This is
my input for rest', 'input 2 Rest')" />
Line 53:
Line 54:                </div>
Line 55:          </div>
Line 56:
Line 57:
Line 58:          <div class="tab-pane" style="display : none"
id="pane2">
Line 59:                      This tab is yet to be implemented!
Line 60:          </div>
Line 61:    </div>
Line 62: <p>
Line 63: And now that you are done wit this page, <a href="./loading-
scripts.html">go do this one</a>
Line 64: </p>
Line 65: <script src="./scripts/common-1.js"
type="text/javascript"></script>
Line 66: <script src="/scripts/common-2.js"
type="text/javascript"></script>
Line 67: <script src="../../scripts/common-3.js"
type="text/javascript"></script>
Line 68: </body>
Line 69: </html>
```

C.2. loading-scripts.html

```
Line 1: <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
Line 2:   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
Line 3: <html xmlns="http://www.w3.org/1999/xhtml">
Line 4: <head><title>Loading Scripts</title>
Line 5: <link rel="stylesheet"
Line 6:       href="./css/style.css"
Line 7:       type="text/css"/>
Line 8: <script src="./scripts/fish.js"
Line 9:         type="text/javascript"></script>
Line 10: </head>
Line 11: <body>
Line 12: <div align="center">
Line 13: <table border="5">
Line 14:   <tr><th class="title">Loading Scripts</th></tr>
```

```
Line 15: </table>
Line 16: <p/>
Line 17: <fieldset>
Line 18:   <legend>Invoking Function from Button</legend>
Line 19:   <form action="#">
Line 20:     <input type="button" value="How Much Did You Win?"
Line 21:            onclick='showWinnings1()'/>
Line 22:   </form>
Line 23:   <p id='phish'></p>
Line 24: </fieldset>
Line 25: <p/>
Line 26: <fieldset>
Line 27:   <legend>Invoking Function from <code>script</code>
Tag</legend>
Line 28:   <script type="text/javascript">showWinnings2()</script>
Line 29: </fieldset>
Line 30: <p/>
Line 31: </div></body></html>
```

C.3. css\styles.css

```
Line 1: body { background-color: #fdf5e6 }
Line 2: a:hover { color: red }
Line 3:
Line 4: th.title   { background-color: #ef8429;
Line 5:              font-size: 28px;
Line 6:              font-family: Arial, Helvetica, sans-serif;
Line 7: }
Line 8: th { background-color: #ef8429;
Line 9:      font-size: 18px;
Line 10:      font-family: Arial, Helvetica, sans-serif;
Line 11: }
Line 12: td { font-size: 18px;
Line 13:      font-family: Arial, Helvetica, sans-serif;
Line 14: }
Line 15: legend {
Line 16:   font-weight: bold;
Line 17:   color: black;
Line 18:   background-color: #eeeeee;
Line 19:   border: 1px solid #999999;
Line 20:   padding: 3px 2px;
Line 21: }
Line 22: .ajaxResult { color: #440000;
Line 23:               font-weight: bold;
Line 24:               font-size: 18px;
Line 25:               font-family: Arial, Helvetica, sans-serif;
Line 26: }
Line 27: h2 { color: #440000;
Line 28:     font-weight: bold;
Line 29:     font-size: 18px;
Line 30:     font-family: Arial, Helvetica, sans-serif;
Line 31: }
Line 32: .error { background-color: yellow;
Line 33:         color: red;
Line 34:         font-weight: bold;
Line 35:         font-size: 20px;
```

```
Line 36:         font-family: Arial, Helvetica, sans-serif;
Line 37:         border-style: inset;
Line 38: }
```

C.4. css\styles1.css

```
Line 1: @import url("./css/styles2.css");
Line 2: * {
Line 3:     margin: 0;
Line 4:     padding: 0;
Line 5: }
Line 6:
Line 7: body {
Line 8:     background-color: #D3D7FF;
Line 9: }
Line 10:
Line 11: a:hover {
Line 12:    color: red
Line 13: }
Line 14:
Line 15: header {
Line 16:    font: 26px Verdana regular;
Line 17:    background-color: #69C1EB;
Line 18:    color: white;
Line 19:    padding: 5px 0 5px 3%;
Line 20:    border-top: 3px solid #1289C0;
Line 21:    border-bottom: 3px solid #1289C0;
Line 22: }
Line 23:
Line 24: .mainContainer {
Line 25:
Line 26:    margin: 0 auto;
Line 27:    width: 50%;
Line 28:    border: 3px solid white;
Line 29: }
Line 30:
Line 31: .form-container {
Line 32:    padding: 3%;
Line 33: }
Line 34:
Line 35: .result-box {
Line 36:    position: relative;
Line 37:    height: 100px;
Line 38:    border: 1px solid white;
Line 39:    padding: 2%;
Line 40: }
Line 41: .right-pane-no-show{
Line 42:    border-bottom:2px solid white;
Line 43: }
Line 44: .left-pane-no-show{
Line 45:    border-bottom:2px solid white;
Line 46: }
Line 47: .tab-pane {
Line 48:    border-radius: 15px;
Line 49:    padding: 2%;
Line 50: }
```

```
Line 51:
Line 52:
Line 53: .tab-button:hover{
Line 54:    cursor:pointer;
Line 55:
Line 56: }
Line 57:
Line 58: .ajaxResult {
Line 59:    color: #440000;
Line 60:    font-weight: bold;
Line 61:    font-size: 18px;
Line 62:    font-family: Arial, Helvetica, sans-serif;
Line 63: }
Line 64:
Line 65: h2 {
Line 66:    color: #440000;
Line 67:    font-weight: bold;
Line 68:    font-size: 18px;
Line 69:    font-family: Arial, Helvetica, sans-serif;
Line 70: }
Line 71:
Line 72: h3 {
Line 73:    color: #440000;
Line 74:    font-weight: bold;
Line 75:    font-size: 14px;
Line 76:    font-family: Arial, Helvetica, sans-serif;
Line 77:    padding: 3px;
Line 78: }
Line 79:
Line 80: .error {
Line 81:    background-color: yellow;
Line 82:    color: red;
Line 83:    font-weight: bold;
Line 84:    font-size: 20px;
Line 85:    font-family: Arial, Helvetica, sans-serif;
Line 86:    border-style: inset;
Line 87: }
Line 88: #pane2{
Line 89:    height:600px;
Line 90:    text-align:center;
Line 91: }
```

C.5. css\styles2.css

```
Line 1: .tab-button{
Line 2:     width: 50%;
Line 3:     box-sizing: border-box;
Line 4:     float: left;
Line 5:     padding: 2%;
Line 6:     border-left:2px solid white;
Line 7:     font-weight: bold;
Line 8:     font-size: 18px;
Line 9:     font-family: Arial, Helvetica, sans-serif;
Line 10: }
Line 11: .tab-button-selected{
Line 12:    background-color:#5F789E;
```

```
Line 13: }
```

C.6. scripts\common-1.js

```
Line 1:
Line 2:
Line 3:
Line 4:
Line 5:
Line 6:
Line 7: // This function takes in the id of the pane
Line 8: //result: show the pane whose id is input and hide the other
pane
Line 9: function toggleTab(tabID) {
Line 10:
Line 11:    document.getElementById("pane1").style = "";
Line 12:    document.getElementById("pane2").style = "";
Line 13:    //show pane 1
Line 14:
Line 15:    if(tabID == 'pane1'){
Line 16:
Line 17:          //first hide the other pane
Line 18:
Line 19:          document.getElementById("pane2").style.display =
"none";
Line 20:          document.getElementById("pane2-btn").style.background
= "#5F789E";
Line 21:
Line 22:          //show the wanted pane
Line 23:          document.getElementById("pane1").style.display =
"block";
Line 24:          document.getElementById("pane1-btn").style.background
= "#D3D7FF";
Line 25:    }// show pane 2
Line 26:    else{
Line 27:
Line 28:          //first hide the other pane
Line 29:          document.getElementById("pane2").style.display =
"block";
Line 30:          document.getElementById("pane2-btn").style.background
= "#D3D7FF";
Line 31:
Line 32:          //show the wanted pane
Line 33:          document.getElementById("pane1").style.display =
"none";
Line 34:          document.getElementById("pane1-btn").style.background
= "#5F789E";
Line 35:    }
Line 36: }
Line 37:
Line 38:
Line 39:
Line 40:
```

C.7. scripts\common-2.js

```
Line 1: function getDataFromServlet(input1){
```

```
Line 2:     var result = "The inputs received were:- <br> " ;
Line 3:     result += input1+"<br>";
Line 4:     result += " I am the output from the servlet<br>";
Line 5:     var restResultElt =
document.getElementById("servletResultRContainer");
Line 6:     restResultElt.innerHTML = result;
Line 7:
Line 8: }
Line 9:
Line 10: function getDataFromJsp(){
Line 11:    var result = "The inputs received were:- <br> " ;
Line 12:    result +="I did not need an input<br>";
Line 13:    result += " I am the output of the rest API<br>";
Line 14:    var restResultElt =
document.getElementById("jspResulttContainer");
Line 15:    restResultElt.innerHTML = result;
Line 16:
Line 17: }
```

C.8. scripts\common-3.js

```
Line 1: // This function is called upon RestAPI button click
Line 2: function getDataFromRest(input1, input2){
Line 3:     var result = "The inputs received were:- <br> " ;
Line 4:     result += input1+"<br>";
Line 5:     result += input2+"<br>";
Line 6:     result += " I am the output of the rest API<br>";
Line 7:     var restResultElt =
document.getElementById("restResultContainer");
Line 8:     restResultElt.innerHTML = result;
Line 9:
Line 10: }
```

C.9. scripts\phish.js

```
Line 1: function getMessage() {
Line 2:   var amount = Math.round(Math.random() * 100000);
Line 3:   var message =
Line 4:     "You won $" + amount + "!\n" +
Line 5:     "To collect your winnings, send your credit card\n" +
Line 6:     "and bank details to oil-minister@phisher.com.";
Line 7:   return(message);
Line 8: }
Line 9:
Line 10: function showWinnings1() {
Line 11:    var elem = document.getElementById('fish');
Line 12:    elem.innerHTML=getMessage();
Line 13: }
Line 14:
Line 15: function showWinnings2() {
Line 16:   document.write("<h1><blink>" + getMessage() +
Line 17:                  "</blink></h1>");
Line 18: }
```

C.10. List of defects in the given source code

Bug ID	File Name	Line Num	Description	Other Comments
1	common-2.js	5	Reference to incorrect HTML Dom ID 'servletResultRContainer'	Bug reported by HJCDepend after correcting bug 17
2	common-2.js	6	access to undefined variable 'restResultElt'	Bug reported by HJCDepend after correcting bug 17
3	common-2.js	14	Reference to incorrect HTML Dom ID 'jspResulttContainer'	Bug reported by HJCDepend after correcting bug 17
4	common-2.js	15	access to undefined variable 'restResultElt'	Bug reported by HJCDepend after correcting bug 17
5	Index.html	67	Incorrect path to common-3.js	
6	Index.html	52	undefined function call 'getDataFromRest' becausecommon-3.js path issue	
7	Index.html	19	invalid argument count to toggletab	
8	Index.html	18	call to undefined function 'togglextab' because typo	
9	Index.html	29	Unavailable css class 'result'	
10	loading-scripts.html	8	Incorrect path to phish.js	
11	loading-scripts.html	6	Incorrect path style.css	
12	loading-scripts.html	21	undefined function call 'showWinnings1' because script path is wrong	Bug reported by HJCDepend after correcting bug 10
13	loading-scripts.html	14	class title not found in css	
14	loading-scripts.html	11	call to unavailable DOM ID fish	Bug reported by HJCDepend after correcting bug 10
15	loading-scripts.html	12	access to undefined variable 'elem'	Bug reported by HJCDepend after correcting bug 10
16	styles1.css	1	Incorrect path to styles2.css	
17	Index.html	66	Incorrect path to common-	

			2.js	
18	Index.html	31	getdataFromJsp undefined	
19	Index.html	42	getDataFromServlet is undefined	
20	loading-scripts.html	28	Call to undefined function showWinnigs2	Not reported by HJCDepend

C.11. Task 1 responses summary

Dependency type	Bug ID	Number of Group A Responses	Number of Group B Responses
HTML-to-JS	5	6	11
	6	2	10
	7	1	8
	8	5	11
	10	8	5
	12	2	4
	17	3	9
	18	2	10
	19	1	10
HTML-to-CSS	9	1	10
	11	2	5
	13	0	4
JS-to-HTML	1	3	1
	2	0	0
	3	2	1
	4	0	0
	14	0	0
	15	0	0
JS-to-JS	20	5	0
CSS-to-CSS	16	2	11

	Total	45	110

C.12. Task 2 responses summary

Bug ID	Number of Group A Responses	Number of Group B Responses	Average Time By Group A	Average Time By Group B
1	4	7	53	48
2	0	3	0	85
3	3	8	78	65
4	0	2	0	10
5	7	11	71	29
6	0	0	0	0
7	2	7	19	30
8	4	10	30	256
9	2	6	22	52
10	6	4	55	30
11	2	7	32	29
12	1	0	10	0
13	1	0	41	0
14	3	5	36	28
15	0	0	0	0
16	2	10	28	26
17	5	11	64	36
18	0	0	0	0
19	0	1	0	30
20	1	0	48	0

C.13 Summary of defects by participants in Group A

Unique user ID Group A	Total Responses	Invalid Responses	Precision (%)	Recall (%)
10lr0	8	2	75	30
1r49n	8	3	62.5	25
cczhi	6	0	100	30
f6ear	7	0	100	35
H47jp	6	2	66.66666667	20
h5dz6	5	1	80	20
i26yj	5	1	80	20

lr01r	3	1	66.66666667	10
lyk308	3	0	100	15
pqeb0	3	0	100	15
sr8y4	2	1	50	5
Total	56	11		

C.14 Summary of defects by participants in Group B

Unique user ID Group B	Total Responses	Invalid Responses	Precision (%)	Recall (%)
1205959296	8	1	87.5	35
7pija	14	1	92.85714286	65
7vzmk	7	0	100	35
8kstn	10	0	100	50
90oyp	9	0	100	45
ffbt9	12	0	100	60
foapx	10	0	100	50
j2xwy	9	0	100	45
ulwor	12	0	100	60
yd7iu	9	0	100	45
z194s	12	0	100	60
Total	112	2		

C.15 Summary of average time taken by participants to find defects.

Dependency type	Bug ID	Average Time By Group A	Average Time By Group B
HTML-to-JS	5	146	21
	6	96	67
	7	71	195
	8	127	23
	10	56	48
	12	57	12
	17	143	243
	18	28	53

	19	54	69
HTML-to-CSS	9	171	37
	11	32	36
	13	0	10
JS-to-HTML	1	38	27
	2	0	0
	3	36	11
	4	0	0
	14	0	0
	15	0	0
JS-to-JS	20	140	0
CSS-to-CSS	16	142	47